WHITEPAPER

Database DevOps

6 tips for achieving continuous delivery

redgate

> "Continuous delivery is the ability to get changes of all types – including new features, configuration changes, bug fixes and experiments – into production, or into the hands of users, *safely* and *quickly* in a *sustainable* way."

Jez Humble, co-author, 'Continuous Delivery'

Contents

Why continuous delivery?

Continuous delivery has changed the rules of the game. It's not just about moving from big bang releases to smaller, faster, more frequent releases. It's about putting in place a process so that reliable software can be deployed without problems, at any time.

One of the biggest advantages is in automating the repetitive development and testing processes that development teams use to deliver, manage, and maintain applications and databases. From version controlling changes to deploying them to different environments, and, when ready, choosing to deploy to production, continuous delivery helps teams reduce risk and increase both efficiency and reliability in the software release process.

More and more companies are adopting it and if you're having problems introducing it, don't worry: that's normal. Fortunately, you're coming to it at the point when all of the hurdles have been faced – and overcome – before.

Importantly, the biggest roadblock to continuous delivery isn't the hardware or software you require, it's the development practices and strategies that need to change to accommodate it.

Those practices, remember, also include database development. Redgate's 2018 State of Database DevOps Survey revealed that 76% of developers are responsible for both database and application development, and 35% of teams deploy database changes either daily or more than once a week.

Tellingly, it also showed that the greatest drawbacks of traditional siloed database development are the increased risk of failed deployments, slower development and release cycles, and an inability to respond quickly to changing business requirements.

So not including database development in continuous delivery can hinder and even prevent companies and organizations gaining the advantages it promises. With that in mind, the following pages give six pointers for starting or enhancing your continuous delivery journey.

Tip 1: Version control doesn't stop at the application

Version controlling application code is becoming standard practice, and Redgate's 2018 State of Database DevOps Survey revealed 81% of companies and organizations already use it. The single source of truth it provides results in an auditable trail of who made what changes and when, encourages collaboration, and means a common version exists to revert back to if there's a problem.

For continuous delivery to work effectively, however, version control needs to go beyond the application and encompass every element in the development process:

- **Databases**
  The code behind database schemas is just that: code. There are tools out there that can version control schemas and reference data, and plug into and integrate with the same systems used for applications. They can also maintain the referential integrity of your database, let you roll back any changes you don't want, and avoid losing data during deployments.

- **Configuration**
  It also makes sense to version control elements such as server configuration properties, network configuration scripts, database server settings and scripts to define database users and roles, and their permissions.

- **Other content**
  Add related content like EULA documentation, website images and deployment scripts, and version control becomes the one source of truth for everything connected to your application.

Tip 2: If you're in a team, branching helps develop further, faster

By its very nature, continuous delivery speeds up software development. This, in turn, encourages development teams to start working on several projects at the same time. While there are bug fixes and quick wins going on over here, new features can be developed over there.

All of which means branching. It can be regarded as a pain, but it's advisable if you want to practise true continuous delivery and be able to release at any time, all of the time.

There is a caveat, however, because as Jez Humble points out on his Continuous Delivery blog, *The more work you do on your branch, the more likely it is you will break the system when you merge into mainline.* The key is to have one main branch that is always releasable, and merge little and often from the other development branches.

This is supported by the 2017 State of DevOps Report from Puppet and DORA which, while strongly advocating branching, comments:

> *"**High performers** have the shortest integration times and branch lifetimes, with branch life and integration typically lasting hours. **Low performers** have the longest integration times and branch lifetimes, with branch life and integration typically lasting days."*

Collaboration is also key. If everyone in the team knows what everyone else is working on, and roughly when they plan to release, conflicts will be avoided later.

Tip 3: Writing tests takes time – but saves more time

It may seem like a paradox, but the time and effort spent writing tests is repaid several times over. Deployment errors reduce, as does the requirement for bug fixes. Users are happier and more engaged. Code becomes robust and maintainable, which further decreases potential issues in the future.

While writing tests quickly and with good coverage can be hard work, automating them in a continuous integration process brings many advantages. As soon as changes are committed to version control, they trigger a build which tests them and provides instant feedback if the build fails. Errors are caught sooner, earlier in the pipeline, and can be fixed while the code is still fresh in the developer's mind.

It works too, and is specifically called out as a key aspect of continuous delivery in the previously mentioned 2017 State of DevOps Report:

> *"We found that the following all positively affect continuous delivery: comprehensive use of version control; continuous integration and trunk-based development; integrating security into software delivery work; and the use of test and deployment automation. Of these, test automation is the biggest contributor."*

The key to writing good tests is to start with the most critical parts of the application, and then move to the less critical parts. And don't worry whether it's a unit test or an integration test. The lines can sometimes blur and the important thing is to have good coverage and tests that don't take an age to run.

Tip 4: NuGet packages make life a lot easier

NuGet packages are on the rise. Not just because of the number that are available (over 100,000 unique packages, with over 6.5 billion downloads and counting), but because the NuGet extension for Visual Studio is a free and powerful package manager that makes using those packages simple.

Rather than downloading, integrating and configuring an open source component into your project, NuGet takes care of it for you, as well as updating the config file if required, and providing the option to update components and remove them if required.

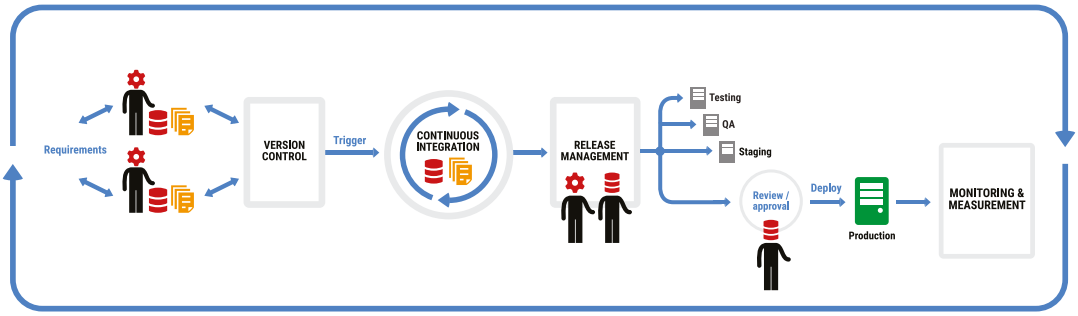
Because NuGet packages are integrated into Visual Studio, it's easy to add or update them in your project. You also don't have to commit the binaries because NuGet can download packages using just the name and version number.

That's one side of the coin. The flipside is that you can you create your own libraries as NuGet packages and publish them to private package repositories. This will improve build times because you're depending on binaries rather than source, and help separate concerns and reduce variability.

Tip 5: Automate, automate, automate

Not everything can be automated, but where it can be, it makes sense because it smooths the development process and highlights errors at earlier stage.

As can be seen in the diagram below, the key to introducing automation is version control, which makes continuous integration possible. Once in place, every time a change is committed to version control, it triggers an automatic build process that tests the change and flags up any errors in the code.



It also opens the door to automating further stages in the development process for both the application and the database, turning it into a reliable, repeatable workflow. Key considerations are:

- **Automate build and test runs**
  There are many continuous integration systems such as TeamCity and Jenkins that will produce an artifact ready for release.

- **Running a build isn't just for applications**
  Anything that needs to be consumed somewhere else should be built, so include libraries and databases too, preferably using tools that plug into the same infrastructure.

- **Automate releases and deployments**
  Writing scripts and running them from your build system is fine for simple projects, but in most cases a release management system like Octopus Deploy will be a much better fit for your environments, team and processes. Once again, database deployments can be integrated into the same system.

Tip 6: Start small and others will follow

In a perfect world, you could start following all of the tips immediately. But you're already busy, you'll have projects on the go, and getting buy-in from management is sometimes hard.

So the final tip is to identify just one improvement to your process that you'd like to try, and tell others about it. Work on something small, prove that it helps, and you'll convince others to follow.

Further resources

However far along you are on your journey to continuous delivery, you may also find the following resources useful.

- **The 2018 State of Database DevOps**
  The latest annual survey of SQL Server database professionals reveals how many are introducing the various stages of continuous delivery, how it's changed over the last 12 months, and what the key challenges are.

- **Continuous Delivery – how do application and database development really compare?**
  A summary of the 2018 State of Database DevOps Survey, looking into how – and why – the adoption rates of version control, continuous integration, automated testing, and automated deployments vary between applications and databases.

- **Bringing DevOps to the database. Part 1 and Part 2**
  Two articles that introduce the stages of continuous delivery that enable database DevOps. Part 1 covers version control, and Part 2 demonstrates how this can be followed by continuous integration and release management.